定積分計算におけるURRの有用性について(数値解析の基礎理論とその周辺)

AUTHOR(S):

浜田, 穂積

CITATION:

浜田, 穂積. 定積分計算におけるURRの有用性について(数値解析の基礎理論とその周辺). 数理解析研究所講究録 1988, 676: 320-329

ISSUE DATE:

1988-12

URL:

http://hdl.handle.net/2433/100954

数理解析研究所講究録
676巻 1988年 320-329

定積分計算におけるURRの有用性について

日立・中研　　浜田　穂積 (HAMADA, Hozumi)

1. はじめに

計算機内部で実数値を表現し計算するために，浮動小数点表現法が用いられてきた．さきに現行の浮動小数点表現法の欠点を指摘し，ほとんど全ての欠点を克服する新い数値表現法を考案した．URR(Universal Representation of Real Numbers) と呼ばれるこの数値表現法は，

(1) オーバーフローが事実上起らない．

(2) 長さの異なるデータ間で完全互換性がある．

という主要な特長の外，数値表現法に望まれる多くの好ましい特性を備えている．

ところでオーバーフローは，起ると厄介であることは確かであるが，実務的には，素朴に考えられるよりは，かなり発生する確率が低くなっているため，現行システムとの互換性を損なってまで規格を変更しにくいと思われている．実務的にあまりオーバーフローの起らない理由は，経験を積んだプログラマは問題に個別の工夫によって避けてきている，あるいはオーバーフローそのものが避けられないときは，解法を変更して逃げているからである．この様な理由で，URRを未だ計算機に採用する段階に至っていない．ただ，これには以下のような楽観的な誤解がある．

(a) どんな問題でもオーバーフローを回避する工夫が可能である．

(b) オーバーフローするのは答が誤った方向に進んでいるからであり，正しい計算ではオーバーフローすることはない．

(c) 計算結果は，単精度計算では約7桁，倍精度計算では約16桁の有効数字が常に得られる．URRでは場合によって精度が低下することがある．

これ等はすべて誤りであるが，数値解析の非専門家には理解し難いことである．この様な理由でURRが絶対必要だという説得材料を探していた．本年2月下旬に東大でURRに関するシンポジウムが2日間にわたって開かれた．その会で森正武・筑波大学教授が，自身が考案した二重指数積分法とURRが相性が良いと思うと発言し，後刻定積分の数値計算例を送付して下さった．その計算を行ってみると，Mシリーズでは（とても解として採用できない様な）かなり大きい打切り誤差がでたが，URRによると一般の応用計算で期待できるのと同程度の丸め誤差で解が得られた．またIEEE標準でも計算してみたが，両者の中間の結果であった．その分析結果を報告する．

2. URRによる定積分の計算

2. 1 積分問題

森教授から提供された問題は次の通りである．

(i) $\int_{-1}^{1} \frac{dx}{(1-x)^{\alpha}}$ を，$\alpha=0.5(0.1)0.9(0.01)0.95$ について求める．

(ii) $\int_{0}^{\infty} \frac{dx}{(1+x)^{\alpha}}$ を，$\alpha=1.02(0.02)1.2$ について求める．

しかしここでは，URRの必然性を説明容易にするため，少し問題を変形した．

$\int \frac{dx}{x}=\log|x|$ であるから，$\int_{0}^{1}\frac{dx}{x}$ も $\int_{1}^{\infty}\frac{dx}{x}$ も有界でない．そこでxをx^{α}で置き換えてみる．

(i) $\alpha<1$ のとき $\int_{0}^{1} \frac{dx}{x^{\alpha}} =\frac{1}{1-\alpha}$

(ii) $\alpha>1$ のとき $\int_{1}^{\infty} \frac{dx}{x^{\alpha}} =\frac{1}{\alpha-1}$

である．そこで(i)では$\alpha=0.9(0.01)0.99$について，(ii)では$\alpha=1.01(0.01)1.1$について計算した．計算はすべて64ビット・データ（倍精度）で行った．計算結果から誤差を求めたものを表1に示す．ここでMシリーズとはHITAC Mシリーズの場合で，IBM S/370と同じであり，通常の計算結果である．

上記2つの問題の結果をまとめて示した．IEEE標準はB16のModula-2で計算した．URRとあるのは，日立中研のHITAC M680にインプリメントしたURRエミュレータによる結果である（括弧内の数値の意味は後述）．表1の結果を図示したのが，図1である．

2. 2 誤差の評価

図1を見ると，重要な2点に気付く．まず第一にMシリーズにおけるこの大きい誤差は何

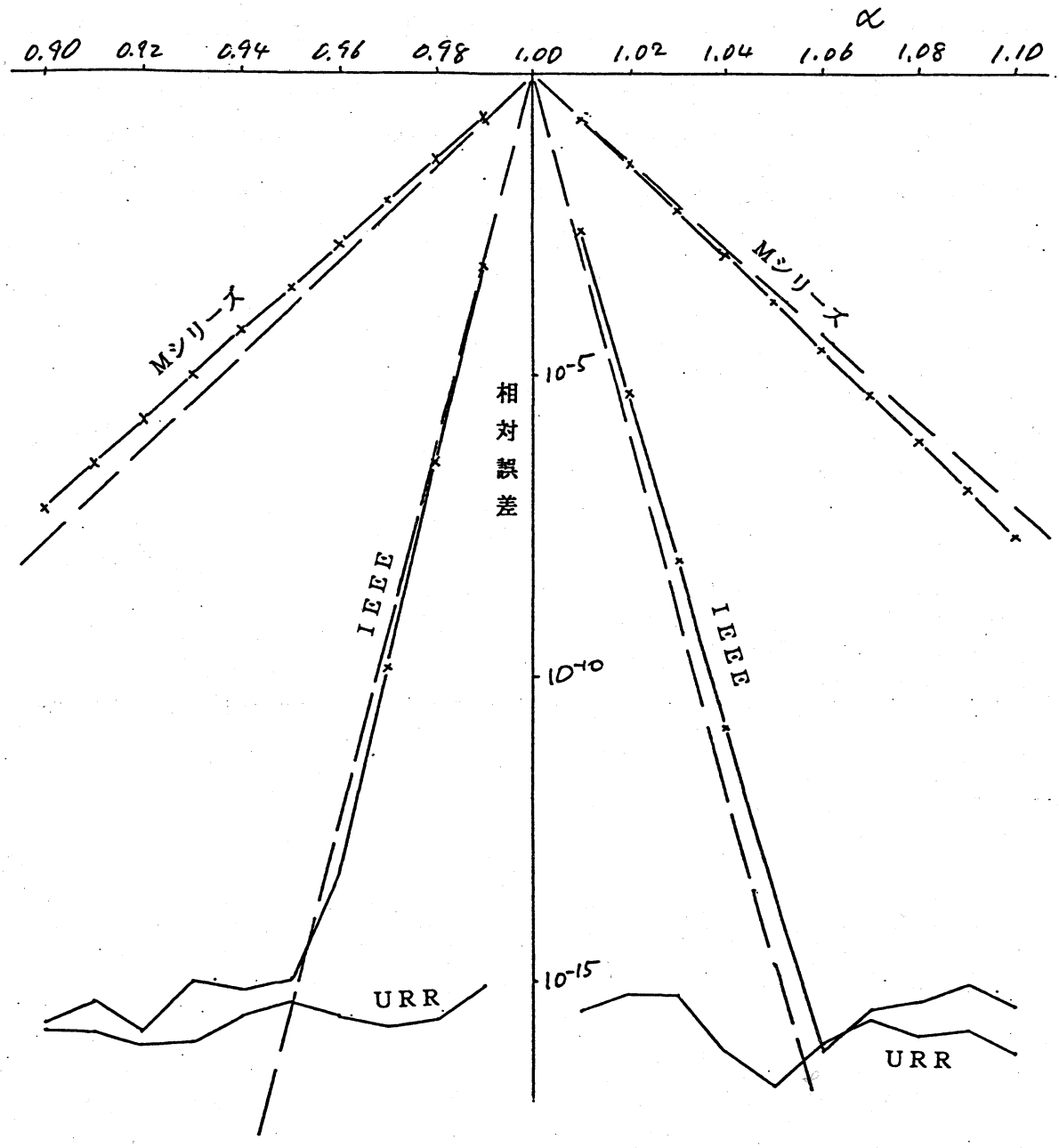

図１．各表現法における積分の誤差
（６４ビット・データ）

故に発生したものか．第二にそれがほとんど直線にのっていることが不思議である．もっともこの種の誤差曲線が大抵の場合打切り誤差を示すことは経験上知られている．この誤差の発生の理由を解明するため，二重指数積分法の指示する変換を行った後の被積分関数の値をプロットしてみた．(i)の場合を図2に示す．この計算はURRによった．αの値が1に近付くにしたがって山が偏って，計算が困難になる様子が見える．xの値と変換前の被積分関数値の何れもがオーバーフローもアンダーフローもしないためのxの値の条件をtに翻訳すると，$-4.72 \leqq t$となる．もちろんtが正の方にも条件が付くが，この場合は問題にする必要はない．同様にIEEEの場合は$-6.11 \leqq t$である．$t=-4.72$と$t=-6.11$の位置に縦に直線を引いてあるが，それぞれこの線より右の部分の面積しか計算できないのであるから，大きい誤差の発生する理由が理解できる．計算過程を振り返って見ればすぐに理解できると思うが，現行計算機で計算可能な条件は(先に述べたことと重複するが)，次の値がオーバーフローもアンダーフローもし

表1．定積分計算の誤差

α	Mシリーズ	IEEE標準	URR
0.90	5.97e-8(1.49e-8)	1.78e-16(1.72e-31)	1.33e-16
0.91	3.32e-7(9.03e-8)	4.44e-16(2.05e-28)	1.33e-16
0.92	1.83e-6(5.48e-7)	1.42e-16(2.44e-25)	8.88e-17
0.93	9.95e-6(3.32e-6)	9.24e-16(2.91e-22)	9.77e-17
0.94	5.37e-5(2.01e-5)	7.11e-16(3.47e-19)	2.49e-16
0.95	2.86e-4(1.22e-4)	1.07e-15(4.14e-16)	4.44e-17
0.96	1.50e-3(7.40e-4)	6.05e-14(4.94e-13)	2.66e-16
0.97	7.79e-3(4.51e-3)	1.53e-10(5.89e-10)	1.78e-16
0.98	3.98e-2(2.83e-2)	3.53e-7 (7.03e-7)	2.22e-16
0.99	2.01e-1(1.71e-1)	6.76e-4 (8.38e-4)	7.90e-16
1.00			
1.01	1.83e-1(1.74e-1)	4.37e-4 (8.99e-4)	3.29e-16
1.02	3.29e-2(3.04e-2)	1.44e-7 (9.28e-7)	6.13e-16
1.03	5.84e-3(5.30e-3)	3.88e-11(1.09e-9)	5.42e-16
1.04	1.02e-3(9.77e-4)	9.81e-15(1.47e-12)	7.11e-17
1.05	1.75e-4(1.87e-4)	5.33e-16(2.24e-15)	1.78e-17
1.06	2.97e-5(3.71e-5)	4.97e-16(3.85e-18)	8.88e-17
1.07	4.97e-6(7.58e-6)	3.20e-16(7.47e-21)	2.22e-16
1.08	8.22e-7(1.59e-6)	4.26e-16(1.63e-23)	1.24e-16
1.09	1.34e-7(3.45e-7)	8.35e-16(3.96e-26)	1.51e-16
1.10	2.18e-8(7.67e-8)	3.55e-16(1.08e-28)	6.22e-17

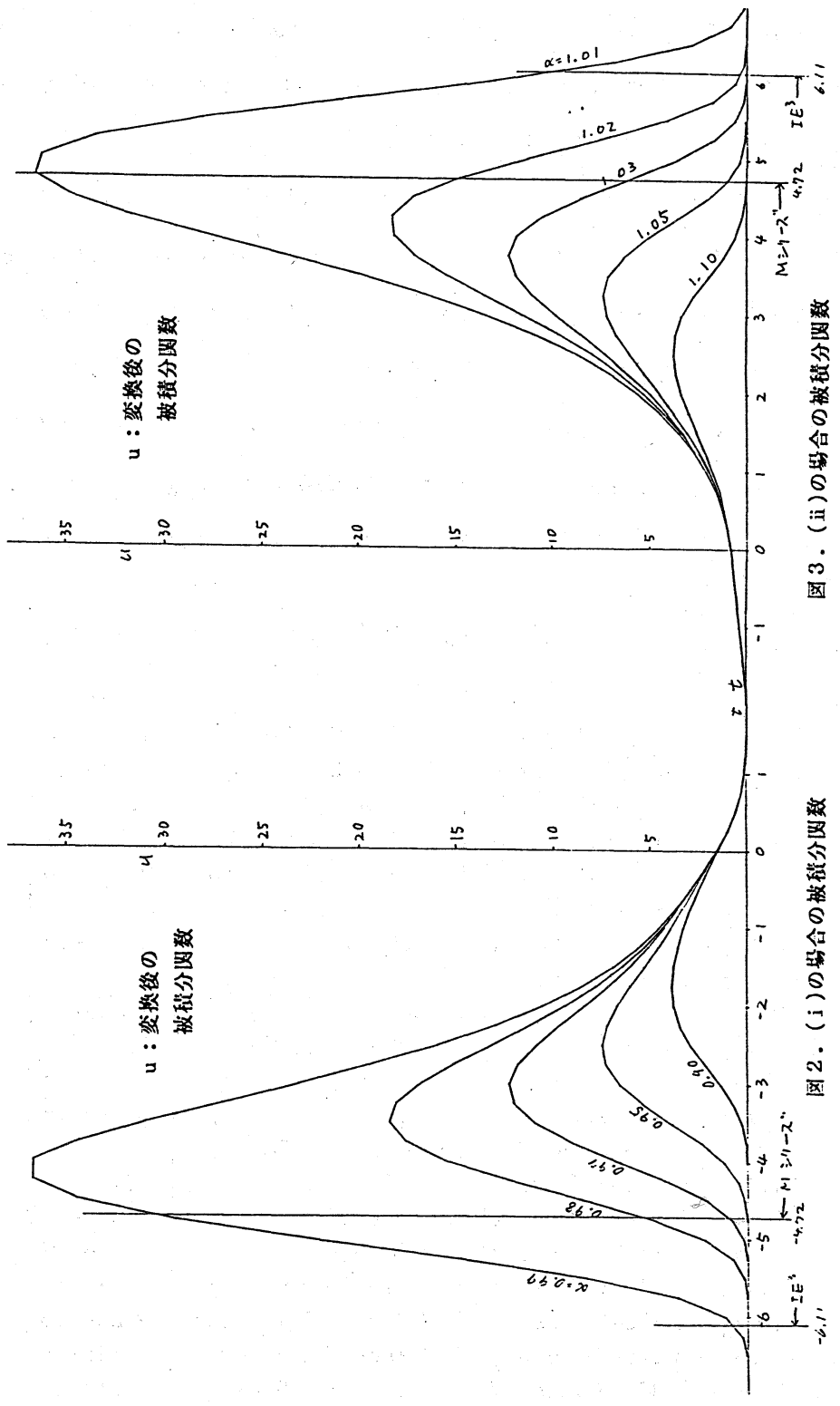

図３．(ii)の場合の被積分関数

図２．(i)の場合の被積分関数

ないことである．変換関数を $x=\phi(t)$ とするとき，

(a) 独立変数 x

(b) 変換前の被積分関数 f(x)

(c) 変換関数の導関数 $\phi'(t)$

である．もちろん

(d) 変換後の被積分関数 $f(\phi(t))\phi'(t)$

という条件も付くが，二重指数積分法による場合は問題にしなくてもよい．実際図2ではオーバーフロー・アンダーフローには無縁に見えるが，(d) を得る以前に (b)，(c) の過程を通らねばならないことに注意する必要がある．同様に (ii) の場合を図3に示した．ある特定の被積分関数の場合は，(d) を直接計算してオーバーフロー・アンダーフローの問題を避けることができるかもしれない．しかしこの様にできるのは，被積分関数固有の性質を用いなければ無理で，一般の関数に適用できる方法はない．

ところで，上述の誤差は二重指数積分法が生み出しているのではないかという疑問が出るかも知れない．しかし上で述べた計算法を考察して見れば，変換前の制約 (a)，(b) が未だ残っていて，それによって縛られているのであって，積分法は，オーバーフロー・アンダーフローに関して本質的な役割を果たしていないことが分る．二重指数積分法は，積分区間が無限大であるとか，被積分関数値が扱い兼ねるほど大きくなって計算が困難であるものを，能率的に高精度で計算できる方法といえる．また，変換後の被積分関数を何処から何処まで積分すれば，実用的な精度で解が得られるかの情報も与えてくれる．

そこで，上の (a)，(b)，(c) の条件をすべて満たす x の範囲を定めて，その区間の定積分を計算し，相対誤差を計算した．これを表1のカッコ内と図1の破線で示した．これを理論誤差と呼ぶことにする．二重指数積分法による計算誤差とよく一致することが分ると思う．

一方 URR の場合は，オーバーフロー・アンダーフローが無いから，x の制約がなく，積分区間を限る必要が無いので，丸め誤差の程度の解が得られていて，これが本来の姿である．

2.3 アンダーフローの問題

その後，千葉大学の小野令美講師から，誤差関数の計算において，級数とか連分数によれば関数値が正しく計算できるのに，二重指数積分法ではアーギュメントが大になると誤差が急激

に増加するがなぜですか，という質問を受けた．結論から先に言うと，これはアンダーフローの処理として値を０にすることから起るトラブルである．先の例では (a) あるいは (b) がオーバーフローして，必要な所までの和を取れないために起る打切り誤差が発生する現象であったが，この場合は，結果としての積分の値が極端に小さいとき (この値自体はアンダーフローしないとしても)，(b) あるいは (c)，あるいはそれ等の積としての (d) がアンダーフローして，機械の処理として０にされるため，本来有限の値が加え合わせられねばならないのに，実質的に多くの項が加えられないために起る誤差であることが分った．しかも，大抵のコンパイラでは，オーバーフローの場合と違ってアンダーフローの場合は特別の注意を喚起することなく，だまって結果を０にするので，誤差の大きい計算結果を得ても気付かないままになるから，事態は深刻であると言えよう．アンダーフローしたとき値を０にすることは「罪」ではないと思われているようであるが，そうは言えない一例である．

3. 結果の考察

図１において，Ｍシリーズの結果は打切り誤差であると述べた．打切り誤差は計算の手間と関係し，通常計算の手間を惜しまなければそれだけ誤差を小にできる．数値計算において，誤差が大でなければ手間はかからない方が良いから，打切り誤差が丸め誤差より小になる条件で，最も手間のかからない方法を選ぶ．ところがここでの主題である数値積分の場合は，打切らなければならない要因が機械の仕様にあるから，誤差を小さくするために手間をかけようとしても，それが不可能であることが本質的である．関数値を計算するときに，独立変数の値か関数値がオーバーフローあるいはアンダーフローするため計算できない領域があり，その領域での積分値が無視できないほど大であれば問題になることを示している．この様な問題を回避するためにしばしば用いられる対策としてスケーリング法がある．ところがこの方法は常に可能ではなく，同次式であるとか，極めて厳しい特定の条件を満たす場合だけである．このような工夫は被積分関数個々に行わねばならず，一般論にはなり難い．

IEEE の場合は，Ｍシリーズの場合と比べて表現可能範囲が広い分だけ打切り誤差が小さく，丸め誤差で求められる範囲が広いことが分る．しかしこの場合も，条件が悪ければさらに大きい打切り誤差が生じることは明らかである．

URRの場合は事実上表現可能範囲に限界がないから，打切り誤差が丸め誤差より小になることを確認して計算を打切ることができる．ところで，URRで良い結果が出ている理由は，現行表現法では処理不可能な数値範囲を利用しているからであるが，この範囲では仮数部が短いため現行表現法と比べて精度がかなり減少することが指摘されている．したがって，その効果が出て丸め誤差が大となり，Mシリーズや，IEEEより誤差が大となっても不思議でない筈であるが，実際にはそうなっていない．このことには重要な意味が有ると思う．仮数部が短いのは非日常的な値であり，その場合「自然」はそれの影響の重みを小さく制御するので，大きい誤差の影響を弱めているからではないかと思える．しかし重みを決してゼロにはせず，常に視野には収めているというのがURRの大きい特徴である．

打切り誤差に関して重要な事実をもう一つ指摘しておきたい．計算に用いる精度（単精度とか倍精度のこと）は，丸め誤差には直接影響するが，打切り誤差とは無関係であるということである．したがって，先に述べた定積分の計算結果の誤差を調べる目的で，しばしば簡便法として用いられる次の方法を用いる場合を考えてみる．すなわち例えば倍精度で計算した結果の誤差を調べるために，同じ事を4倍精度でも計算し，その上位何桁が倍精度の結果と一致するかを調べるものである．もし上位16桁が一致すれば，正しく16桁計算できたと結論する．しかしこれは4倍精度の結果が十分正確（少くとも倍精度計算で得られる最小の誤差より小）であることが前提である．現行方式では打切り誤差が支配的なので，倍精度と4倍精度の結果の値は一致するから，「倍精度計算は十分正確に計算できた」と結論する誤りを犯しかねない．一方URRで計算すれば丸め誤差の程度で計算できるので，上述の確認法が誤りなく適用できる．

また，図1の下方に4倍精度の丸め誤差のレベルを想定し，現行方式の打切り誤差の線を延長して交わる点を想定してみれば，現行方式で丸め誤差の程度で計算できる条件は，倍精度より4倍精度の方が狭くなるという，常識に反する（実はよく考えてみるとあたりまえな）結果が得られることが容易に理解できよう．すなわち，ここで取り上げたような定積分の数値計算の場合，現行方式においては，計算精度を向上させようとして倍精度計算から4倍精度計算に変えることは，何等効果が得られないばかりか，誤った判断をさせる可能性がある．

以上の結果をある人に見せたら，次のような批判があった．「確かに$\alpha=0.9\sim1.1$でURRの威力があることは分った．しかし現在行われている数値計算の内で，この条件の数値積分は何

パーセントであろうか．極く極く僅かであるはずだ．そのように僅かな場合に効果があるだけなら，計算機を変更するほどの価値はない．」このような論理は著者の周辺でよく用いられるが，この場合に限れば論理が逆転している．これまでの計算機では計算できなかったから行われることが少かったのであり，多くの場合このような困難に遭遇すると，別の解法で逃げていたのではなかろうか．

また次のような批判もあった．「そのように解析的に正しい解が分っている問題は数値積分によって解を出すことはない．オーバーフローで解が正しく出ないと言っても，大抵の場合工夫によって解決できる．実際の応用の中にある難しい問題が計算できたというのでなければ意味がない．URR が必要だと言う，為にする我田引水の論理だ．」解析的に正しい解が分っている問題を扱うのは，誤差をきちんと把握して，諸々の分析を容易にするためであり，図 2，図 3 の傾向を示すという点で一般性を失わない．多くの積分困難な関数が類似の傾向を示すはずである．難しい問題を取り上げると，誤差の大きさの原因が，数値表現法に有るのか，積分法に有るのか，問題の困難さに有るのか，判定困難になるし，上述の意味での判断の誤りを起すかもしれないからである．また URR を推進する動機の最大の要素の一つは，問題に依存する工夫によることを排し，なるべく一般的な方法で問題を解決したいという点である．このような理由で，上述の批判は正当でないと考えている．

4. おわりに

URR を定積分の数値計算に適用して著しい効果が認められた．特に無限区間にわたるものとか，区間内で被積分関数の値がオーバーフローして，現行計算機では計算できないような場合にも高い精度で計算できる．またアンダーフローする計算結果を 0 にするために極端な精度低下を招く計算においても，高い精度が確保できることを確認できた．特殊関数（例えば積分指数関数）等で積分表示で簡潔に定義されているものが多い．この場合の関数値の計算などにも有効と思われる．

計算機メーカは今まで，たとえば M シリーズで処理可能な実数値（浮動小数点数）の上限約 10^{77} を，われわれが必要とするには十分大きい数値であると考えてきた．またその逆数の下限約 10^{-77} も十分小さい数値であり，それより小さい値が出現することはめったにないが，

もしあったとしても，それを0で置換えていっこうに差支えないと考えてきた．しかしながら今回の結果を見ると，数学で用いる無限大というのは本当に大きい数で，無限区間の幅と比べれば，10^{77} までの幅は事実上0に等しいことが分る．上限値をどんなに大きくとっても，その値を固定する限り同じである．数学が無限大を要求するならそれを実現したい．しかし現実には，表現可能範囲を固定するという方法では不可能である．

この状況については，次のイソップのたとえ話を連想する．狐は，高い木になっている葡萄が欲しいが，取れないことが分っているので「あの葡萄は酢っぱい」という．われわれは，無限大を手に入れることができないので，「10^{77} で十分」と負け惜しみを言っているのではなかろうか．数学も決して無限大を手に入れているわけではない．必要に応じていくらでも大きい数を扱う，という手法で対処している．計算機の場合も，URRを用いればこれは可能である．

ここで言及した数値積分は，筑波大学・森正武教授の示唆による．また問題自体も森教授から提供を受けたものに基いている．森教授に深甚の謝意を表する．また，Mシリーズを用いたURRでの計算に際して，当研究所第八部・菊池純男氏のお世話になったことに感謝する．

参 考 文 献

1) 森正武: 数値解析と複素関数論．筑摩書房・数理科学シリーズ7(July 1975), pp.274

2) 戸田英雄，小野令美: 入門数値計算．オーム社 (Feb. 1983), pp.236

3) 浜田穂積: 二重指数分割に基づくデータ長独立実数値表現法．情報処理学会論文誌, Vol.22, No.6(Nov. 1981), pp.521-526

4) 浜田穂積: 数値表現法URRの評価．コンピュータ・ソフトウェア，Vol.1, No.3(Oct. 1984), pp.241-249